Министерство образования и науки РФ

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

Тульский государственный университет

КАФЕДРА АВТОМАТИКИ И ТЕЛЕМЕХАНИКИ

**ПЕРЕДАЧА ДАННЫХ МЕЖДУ ПРОЦЕССАМИ. ИСПОЛЬЗОВАНИЕ MAILSLOT**

Лабораторная работа № 5

по курсу «Операционные системы»

Вариант № 3

Выполнил: студент группы 220601 \_\_\_\_\_\_\_\_\_ Белым А.А.

(подпись)

Проверил: \_\_\_\_\_\_\_\_\_ Попов А.И.

(подпись)

Тула 2012

# Цель работы

Целью работы состоит в том, чтобы изучить принципы передачи данных между процессами с помощью MailSlot, научиться применять изученные принципы на практике.

# Задание

Написать программы (сервер - клиент), работающие с каналом MailSlot и определяющие его состояние, продемонстрировать работу с ним.

Программы-клиенты рисуют или загружают из файла изображения, программы-сервера получают изображение через широковещательные mailslot’ы, и отображают их.

# Теоретическая справка

Рассмотрим еще один простой способ организации передачи данных между различными процессами, основанный на использовании датаграммных кана-лов Mailslot Почтовые ящики обеспечивают только однонаправленные соеди-нения. Каждый процесс, который создает почтовый ящик, является "сервером почтовых ящиков" (mailslot server). Другие процессы, называемые "клиентами почтовых ящиков" (mailslot clients), посылают сообщения серверу, записывая их в почтовый ящик. Входящие сообщения всегда дописываются в почтовый ящик и сохраняются до тех пор, пока сервер их не прочтет.

Главная особенность каналов Mailslot заключается в том, что они, в отли-чие от других средств передачи данных между процессами позволяют переда-вать данные в широковещательном режиме. Это означает, что на компьютере или в сети могут работать несколько серверных процессов, способных полу-чать сообщения через каналы Mailslot. При этом один клиентский процесс мо-жет посылать сообщения сразу всем этим серверным процессам.

Клиент может посылать сообщения на почтовый ящик, расположенный на том же компьютере, на компьютере в сети, или на все почтовые ящики с одним именем всем компьютерам выбранного домена. При этом широковещательное сообщение, транслируемое по домену, не может быть более 400 байт. В остальных случаях размер сообщения ограничивается только при создании почтового ящика сервером. Почтовые ящики предлагают легкий путь для об-мена короткими сообщениями, позволяя при этом вести передачу и по локаль-ной сети, в том числе и по всему домену.

Программа может создать на компьютере некий объект, собственно и назы-ваемый Mailslot'ом (почтовым ящиком), доступ к которому может получить уже не только она сама, но и другие программы — причем необязательно вы-полняющиеся на том же компьютере. Это значит, что доступ к mailslots можно получить в том числе из сети. Большинство файерволлов при настройках по умолчанию свободно пропускают трафик, если он идет через mailslots, техно-логия эта создавалась для обмена данными между программами, поэтому MailSlots считается будущим троянологии.

Mailslot является псевдофайлом, находящимся в памяти, и следует ис-пользовать стандартные функции для работы с файлами, чтобы получить к нему доступ. Данные в почтовом ящике могут быть в любой форме – их интер-претацией занимается прикладная программа, но их общий объем не должен превышать 64 Кб. Однако, в отличие от дисковых файлов, mailslot’ы являются временными – когда все дескрипторы почтового ящика закрыты, он и все его данные удаляются. Заметим, что все почтовые ящики являются локальными по отношению к создавшему их процессу; процесс не может создать удаленный mailslot.

Все функции почтовых ящиков Mailslot реализованы в DLL клиентской ча-сти подсистемы Win32 - kernel32.dll. Однако имена, задаваемые приложениями, использующими почтовые ящики, определяют системное пространство имен, управляемое драйвером файловой системы именованных каналов (Named Pipes File System, NPFS.sys) и драйвером файловой системы почтовых ящиков (Mail Slots File System, MSFS.sys).

Канал Mailslot создается серверным процессом с помощью специально предназначенной для этого функции CreateMailslot (прототипы см. Приложе-ние). После создания серверный процесс получает идентификатор канала Mailslot. Пользуясь этим идентификатором, сервер может читать сообщения, посылаемые в канал клиентскими процессами. Однако сервер не может выпол-нять над каналом Mailslot операцию записи, так как этот канал предназначен только для односторонней передачи данных - от клиента к серверу.

Прежде чем приступить к работе с каналом Mailslot, клиентский процесс должен его открыть. Для выполнения этой операции следует использовать функцию CreateFile (см. лаб. раб. №5 («Каналы Pipe»)). При этом можно от-крыть канал Mailslot, созданный на другой рабочей станции в сети. Для этого строка имени канала, передаваемая функции CreateFile, должна иметь следую-щий вид:

\\ИмяРабочейСтанции\mailslot\[Путь]ИмяКанала

Можно открыть канал для передачи сообщений всем рабочим станциям за-данного домена. Для этого необходимо задать имя по следующему образцу:

\\ИмяДомена\mailslot\[Путь]ИмяКанала

Для передачи сообщений одновременно всем рабочим станциям сети пер-вичного домена имя задается следующим образом:

\\\*\mailslot\[Путь]ИмяКанала

Напомним, что клиентский процесс может только посылать сообщения в канал Mailslot, но не читать их оттуда. Чтение сообщений из канала Mailslot - задача для серверного процесса.

Запись сообщений в канал Mailslot выполняет клиентский процесс, вызывая для этого функцию WriteFile аналогично записи в обычный файл.

Серверный процесс может читать сообщения из созданного им канала Mailslot при помощи функции ReadFile.

Серверный процесс может определить текущее состояние канала Mailslot по его идентификатору с помощью функции GetMailslotInfo.

С помощью функции SetMailslotInfo серверный процесс может изменить время ожидания для канала Mailslot уже после его создания.

# Инструкция пользователю

Данный набор программ позволяет организовать широковещательную рассылку изображений по сети.

Программа-клиент позволяет загрузить из файла или нарисовать картинку. После чего картинка может быть отправлена в сеть соответствующей кнопкой.

Программа-сервер должна получить список клиентов с помощью соответствующей кнопки. Затем при нажатии кнопки «Получить» она получает сообщения клиентов из сети.

# Инструкция программисту

**Константы:**

const int MAXLEN**=**300**;**

Максимальная длина пакета.

const LPCTSTR GetServerNotify**=**

TEXT**(**"\\\\.\\mailslot\\{ed19aa6c-8132-4a3c-82d7-519145548b39}"**),**

Имя серверного слота для уведомлений о новых клиентах.

PutServerNotify**=**

TEXT**(**"\\\\\*\\mailslot\\{ed19aa6c-8132-4a3c-82d7-519145548b39}"**);**

Имя клиентского слота для уведомлений(для посылки UUID нового клиента).

const int MAXDATA**=**MAXLEN**-sizeof(**PackageType**)-sizeof(**int**),**

Максимальная длина полезных данных в пакете.

MAXINFO**=**MAXLEN**-**MAXDATA**;**

Максимальная длина служебной части пакета.

**Типы данных:**

Перечисление типов пакетов:

enum PackageType**{**

Package\_Head- пакет – заголовок сообщения

Package\_Body – пакет – тело сообщения

Package\_Tail – пакет – конец сообщения

**};**

Структура пакета:

**typedef** struct **{**

PackageType type- тип пакета

int num– номер пакета (для отслеживания дубликатов и потерянных пакетов)

union**{**

int total\_size**;** - общее число байт в сообщении

(для заголовочного пакета)

char data**[**MAXDATA**];** - полезные данные

**};**

**}** Package**;**

Состояние сервера:

**typedef** struct **{**

QList**<**Package**\*>** transmissions**,** - список принятых пакетов

heads**;** - список заголовочных пакетов

QList**<**int**>** lost**,** - список номеров потерявшихся пакетов

tails**;** - список номеров ожидаемых концевых пакетов.

bool first\_run**;** - состояние при первом запуске

int current\_num**;** - номер текущего принятого пакета

**}** ServerState**;**

**Серверная часть.**

bool MakeSlot**(**HANDLE **&**hSlot**,**LPCTSTR lpszSlotName**);**

Создает серверный мэйлслот с указанным именем.

bool ReceiveNotify**(**HANDLE **&**hSlot**,**QUuid **&**uuid**);**

Получает UUID уникального канала клиента из слота для уведомлений.

void OpenUUIDSlot**(**HANDLE **&**hSlot**,**QUuid **&**uuid**);**

Открывает серверный мэйлслот, имя которого задается с помощью UUID.

bool ReceiveData**(**HANDLE hSlot**,** char **\*** **&**data**,** int **&**size**,** ServerState **&**state**);**

Получает данные из серверного слота.

**Клиентская часть.**

bool MakeFile**(**HANDLE **&**hFile**,**LPCTSTR SlotName**);**

Создает клиентский мэйлслот с указанным именем.

void SendNotify**(**HANDLE **&**hSlot**,**QUuid **&**uuid**);**

Отправляет UUID уникального канала в слот для уведомлений.

void GenUUIDSlot**(**HANDLE **&**hSlot**,**QUuid **&**uuid**);**

Открывает клиентсткий мэйлслот, имя которого задается с помощью UUID.

bool TransmitData**(**HANDLE hFile**,** const char **\*** source**,** int size**,** int **&**num**);**

Отправляет данные через mailslot.

# Текст программы

Ниже представлен текст программ для рисования и передачи изображений с использованием MailSlot и написанных на языке C++, в среде Qt Creator 2.5.2 + MinGW-GCC 4.6 с использованием библиотеки Qt.

**Процедуры работы с MailSlot.**

mailslots.h:

#ifndef MAILSLOTS\_H

#define MAILSLOTS\_H

extern "C"**{**

#include "windows.h"

**}**

#include <QUuid>

#include <QList>

const int MAXLEN**=**300**;**

const LPCTSTR GetServerNotify**=**

TEXT**(**"\\\\.\\mailslot\\{ed19aa6c-8132-4a3c-82d7-519145548b39}"**),**

PutServerNotify**=**

TEXT**(**"\\\\\*\\mailslot\\{ed19aa6c-8132-4a3c-82d7-519145548b39}"**);**

enum PackageType**{**

Package\_Head**,**

Package\_Body**,**

Package\_Tail

**};**

const int MAXDATA**=**MAXLEN**-sizeof(**PackageType**)-sizeof(**int**),**

MAXINFO**=**MAXLEN**-**MAXDATA**;**

**typedef** struct **{**

PackageType type**;**

int num**;**

union**{**

int total\_size**;**

char data**[**MAXDATA**];**

**};**

**}** Package**;**

**typedef** struct **{**

QList**<**Package**\*>** transmissions**,**heads**;**

QList**<**int**>** lost**,**tails**;**

bool first\_run**;**

int current\_num**;**

**}** ServerState**;**

bool MakeFile**(**HANDLE **&**hFile**,**LPCTSTR SlotName**);**

bool MakeSlot**(**HANDLE **&**hSlot**,**LPCTSTR lpszSlotName**);**

bool ReceiveNotify**(**HANDLE **&**hSlot**,**QUuid **&**uuid**);**

void SendNotify**(**HANDLE **&**hSlot**,**QUuid **&**uuid**);**

void OpenUUIDSlot**(**HANDLE **&**hSlot**,**QUuid **&**uuid**);**

void GenUUIDSlot**(**HANDLE **&**hSlot**,**QUuid **&**uuid**);**

bool TransmitData**(**HANDLE hFile**,** const char **\*** source**,** int size**,** int **&**num**);**

bool ReceiveData**(**HANDLE hSlot**,** char **\*** **&**data**,** int **&**size**,** ServerState **&**state**);**

#endif // MAILSLOTS\_H

mailslots.cpp:

#include "mailslots.h"

#include <QDebug>

#include <QByteArray>

BOOL WriteSlot**(**HANDLE hSlot**,** void**\*** lpszMessage**,**int size**)**

**{**

BOOL fResult**;**

DWORD cbWritten**;**

fResult **=** WriteFile**(**hSlot**,**

lpszMessage**,**

size**,**

**&**cbWritten**,**

**NULL);**

**if** **(!**fResult**)**

**{**

int e**=**GetLastError**();**

qDebug**()<<**QString**(**"WriteFile failed with %1."**).**arg**(**QString**::**number**(**e**));**

**return** FALSE**;**

**}**

**return** TRUE**;**

**}**

bool TransmitData**(**HANDLE hFile**,** const char **\***source**,** int size**,**int **&**num**){**

int psource**=**0**,**pdata**=**0**;**

Package package**;**

package**.**type**=**Package\_Head**;**

package**.**num**=**num**;**num**++;**

package**.**total\_size**=**size**;**

WriteSlot**(**hFile**,&**package**,**MAXLEN**);**

WriteSlot**(**hFile**,&**package**,**MAXLEN**);**

//WriteSlot(hFile,&package,MAXLEN+1);

**while(**psource**+**MAXDATA**<=**size**){**

package**.**type**=**Package\_Body**;**

package**.**num**=**num**;**num**++;**

**for** **(**pdata**=**0**;**pdata**<**MAXDATA**&&**psource**<**size**;++**pdata**,++**psource**){**

package**.**data**[**pdata**]=**source**[**psource**];**

**}**

WriteSlot**(**hFile**,&**package**,**MAXLEN**);**

//Sleep(10);

WriteSlot**(**hFile**,&**package**,**MAXLEN**);**

//WriteSlot(hFile,&package,MAXLEN+1);

**}**

package**.**type**=**Package\_Tail**;**

package**.**num**=**num**;**num**++;**

**for** **(**pdata**=**0**;**pdata**<**MAXDATA**&&**psource**<**size**;++**pdata**,++**psource**){**

package**.**data**[**pdata**]=**source**[**psource**];**

**}**

WriteSlot**(**hFile**,&**package**,**MAXLEN**);**

WriteSlot**(**hFile**,&**package**,**MAXLEN**);**

//WriteSlot(hFile,&package,MAXLEN);

**return** **true;**

**}**

void GenUUIDSlot**(**HANDLE **&**hSlot**,**QUuid **&**uuid**){**

QString slotname**=**QString**(**"\\\\\*\\mailslot\\%1"**).**arg**(**uuid**.**toString**());**

wchar\_t **\***lpSlotName**=new** wchar\_t**[**slotname**.**length**()];**

slotname**.**toWCharArray**(**lpSlotName**);**

MakeFile**(**hSlot**,**lpSlotName**);**

**delete** lpSlotName**;**

**}**

void OpenUUIDSlot**(**HANDLE **&**hSlot**,**QUuid **&**uuid**){**

QString slotname**=**QString**(**"\\\\.\\mailslot\\%1"**).**arg**(**uuid**.**toString**());**

wchar\_t **\***lpSlotName**=new** wchar\_t**[**slotname**.**length**()];**

slotname**.**toWCharArray**(**lpSlotName**);**

MakeSlot**(**hSlot**,**lpSlotName**);**

**delete** lpSlotName**;**

**}**

void SendNotify**(**HANDLE **&**hSlot**,**QUuid **&**uuid**){**

char **\***data**=**uuid**.**toByteArray**().**data**();**

int size**=**uuid**.**toByteArray**().**size**();**

qDebug**()<<**uuid**.**toString**();**

qDebug**()<<**size**;**

WriteSlot**(**hSlot**,**data**,**size**);**

**}**

bool ReceiveNotify**(**HANDLE **&**hSlot**,**QUuid **&**uuid**){**

DWORD size**;**

DWORD fResult**=**GetMailslotInfo**(** hSlot**,** // mailslot handle

**(**LPDWORD**)** **NULL,** // no maximum message size

**&**size**,** // size of next message

**NULL,** // number of messages

**(**LPDWORD**)** **NULL);**

**if** **(!**fResult**)**

**{**

qDebug**()<<**"GetMailslotInfo failed with "**<<**GetLastError**();**

**return** **false;**

**}**

**if(**size**!=**MAILSLOT\_NO\_MESSAGE**){**

char data**[**size**];**

qDebug**()<<**size**;**

fResult**=**ReadFile**(**hSlot**,**

data**,**

size**,**

**NULL,**

**NULL);**

**if** **(!**fResult**)**

**{**

qDebug**()<<**"ReadFile failed with "**<<**GetLastError**();**

**return** **false;**

**}**

uuid**=**QUuid**(**QByteArray**(**data**,**size**));**

**}** **else**

**return** **false;**

**return** **true;**

**}**

bool MakeFile**(**HANDLE **&**hFile**,**LPCTSTR SlotName**){**

hFile **=** CreateFile**(**SlotName**,**

GENERIC\_WRITE**,**

FILE\_SHARE\_READ**,**

**NULL,**

OPEN\_EXISTING**,**

FILE\_ATTRIBUTE\_NORMAL**,**

**NULL);**

**if** **(**hFile **==** INVALID\_HANDLE\_VALUE**)**

**{**

qDebug**()<<**QString**(**"CreateFile failed with %1."**).**arg**(**QString**::**number**(**GetLastError**()));**

**return** **false;**

**}**

**return** **true;**

**}**

bool MakeSlot**(**HANDLE **&**hSlot**,**LPCTSTR lpszSlotName**)**

**{**

hSlot **=** CreateMailslot**(**lpszSlotName**,**

0**,** // no maximum message size

MAILSLOT\_WAIT\_FOREVER**,** // no time-out for operations

**NULL);** // default security

**if** **(**hSlot **==** INVALID\_HANDLE\_VALUE**)**

**{**

qDebug**()<<**"CreateMailslot failed with "**<<**GetLastError**();**

**return** **false;**

**}**

**return** **true;**

**}**

bool ReceiveData**(**HANDLE hSlot**,**char **\*** **&**data**,**int **&**size**,**ServerState **&**state**)**

**{**

DWORD cbMessage**,** cMessage**,** cbRead**;**

BOOL fResult**;**

Package**\*** buffer**=NULL;**

QList**<**Package**\*>** **&**transmissions**=**state**.**transmissions**,&**heads**=**state**.**heads**;**

QList**<**int**>** **&**lost**=**state**.**lost**,&**tails**=**state**.**tails**;**

bool **&**first\_run**=**state**.**first\_run**;**

int **&**current\_num**=**state**.**current\_num**;**

int pbuffer**,**pdata**=**0**;**

cbMessage **=** cMessage **=** cbRead **=** 0**;**

fResult **=** GetMailslotInfo**(** hSlot**,** // mailslot handle

**(**LPDWORD**)** **NULL,** // no maximum message size

**&**cbMessage**,** // size of next message

**&**cMessage**,** // number of messages

**(**LPDWORD**)** **NULL);** // no read time-out

**if** **(!**fResult**)**

**{**

qDebug**()<<**"GetMailslotInfo failed with "**<<**GetLastError**();**

**return** **false;**

**}**

**if** **(**cbMessage **==** MAILSLOT\_NO\_MESSAGE**)**

**{**

qDebug**()<<(**"No messages..."**);**

**return** **false;**

**}**

**while** **(**cMessage **!=** 0**)** // retrieve all messages

**{**

buffer**=new** Package**;**

fResult **=** ReadFile**(**hSlot**,**

buffer**,**

cbMessage**,**

**&**cbRead**,**

**NULL);**

**if(**cbRead**!=**MAXLEN**){**

qDebug**(**"OH SHI~!"**);**

**return** **false;**

**}**

**if** **(!**fResult**)**

**{**

qDebug**()<<**"ReadFile failed with "**<<**GetLastError**();**

**return** **false;**

**}**

qDebug**()<<**buffer**->**num**;**

**if(**first\_run**){**

**if** **(**buffer**->**type**==**Package\_Head**){**

current\_num**=**buffer**->**num**;**

first\_run**=false;**

transmissions**<<**buffer**;**

**}** **else** **{**

qDebug**()<<**"Waiting for head packet!"**;**

**delete** buffer**;**

buffer**=NULL;**

**}**

**}** **else** **{**

**if** **(**buffer**->**num**>**current\_num**){**

**for(**int i**=**current\_num**+**1**;**i**<**buffer**->**num**;++**i**){**

qDebug**()<<**"Package "**<<**i**<<**" lost!"**;**

lost**<<**i**;** transmissions**<<NULL;**

**}**

transmissions**<<**buffer**;**

current\_num**=**buffer**->**num**;**

**}** **else** **{**

int i**=**lost**.**indexOf**(**buffer**->**num**);**

**if(**i**!=-**1**){**

qDebug**()<<**"Lost packet arrived!"**;**

lost**.**removeAt**(**i**);**

transmissions**[**i**]=**buffer**;**

**}** **else** **{**

qDebug**()<<**"Duplicate arrived!"**;**

**delete** buffer**;**

buffer**=NULL;**

**}**

**}**

**}**

**if(**buffer**!=NULL){**

**switch(**buffer**->**type**){**

**case** Package\_Head**:{**

heads**<<**buffer**;**

int temp**=**buffer**->**num**+**buffer**->**total\_size**/**MAXDATA**+(**buffer**->**total\_size**%**MAXDATA**?**1**:**0**);**

qDebug**()<<**"Expecting tail at "**<<**temp**;**

tails**<<**temp**;**

**}**

**break;**

**case** Package\_Tail**:{**

qDebug**()<<**"Tail received!"**;**

int index**=**tails**.**indexOf**(**buffer**->**num**);**

**if(**index**!=-**1**){**

Package **\***head**=**heads**.**at**(**index**);**

bool ok**=true;**

size**=**head**->**total\_size**;**

data**=new** char**[**size**];**

heads**.**removeAt**(**index**);**tails**.**removeAt**(**index**);**

int start**=**transmissions**.**indexOf**(**head**);**

**for(**int i**=**start**+**1**;**i**<=**start**+**buffer**->**num**-**head**->**num**;++**i**){**

qDebug**()<<**i**;**

Package**\*** temp**=**transmissions**.**at**(**i**);**

**if(**temp**==NULL){**

qDebug**()<<**"There are lost packages!"**;**

ok**=false;**

**}**

**if(**ok**){**

**for** **(**pbuffer**=**0**;**pbuffer**<**MAXDATA**&&**pdata**<**size**;++**pdata**,++**pbuffer**){**

//qDebug()<<pdata;

data**[**pdata**]=**temp**->**data**[**pbuffer**];**

**}**

**delete** temp**;**

**}** **else** **{**

lost**.**removeOne**(**head**->**num**+**i**-**start**);**

**}**

**}**

**for(**int i**=**start**;**i**<=**start**+**buffer**->**num**-**head**->**num**;++**i**){**

transmissions**.**removeAt**(**start**);**

**}**

**if(**ok**){**

qDebug**()<<**"Message received!"**;**

**return** **true;**

**}** **else** **{**

**delete** data**;**

size**=**0**;**

**return** **false;**

**}**

**}**

**}**

**break;**

**case** Package\_Body**:**

**break;**

**default:**

qDebug**()<<**"Error! Unknown packet type!"**;**

**}**

**}**

fResult **=** GetMailslotInfo**(** hSlot**,** // mailslot handle

**(**LPDWORD**)** **NULL,** // no maximum message size

**&**cbMessage**,** // size of next message

**&**cMessage**,** // number of messages

**(**LPDWORD**)** **NULL);** // no read time-out

**if** **(!**fResult**)**

**{**

qDebug**()<<**"GetMailslotInfo failed with "**<<**GetLastError**();**

**return** **false;**

**}**

**}**

**return** **false;**

**}**

**Серверное приложение.**

mainwindow.h:

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QGraphicsScene>

#include <QUuid>

#include <QList>

#include "../Lab5OS/mailslots.h"

**namespace** Ui **{**

class MainWindow**;**

**}**

class MainWindow **:** public QMainWindow

**{**

Q\_OBJECT

public**:**

explicit MainWindow**(**QWidget **\***parent **=** 0**);**

**~**MainWindow**();**

private slots**:**

void on\_action\_receive\_triggered**();**

void on\_action\_clients\_triggered**();**

void on\_tabWidget\_tabCloseRequested**(**int index**);**

private**:**

Ui**::**MainWindow **\***ui**;**

QList**<**QGraphicsScene**\*>** scenes**;**

HANDLE hNotify**;**

QList**<**HANDLE**>** hSlots**;**

QList**<**QUuid**>** uuids**;**

QList**<**ServerState**\*>** states**;**

**};**

#endif // MAINWINDOW\_H

mainwindow.cpp:

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QBuffer>

#include <QDebug>

#include <QGraphicsView>

MainWindow**::**MainWindow**(**QWidget **\***parent**)** **:**

QMainWindow**(**parent**),**

ui**(new** Ui**::**MainWindow**)**

**{**

ui**->**setupUi**(this);**

MakeSlot**(**hNotify**,**GetServerNotify**);**

**}**

MainWindow**::~**MainWindow**()**

**{**

**delete** ui**;**

**}**

void MainWindow**::**on\_action\_receive\_triggered**()**

**{**

char **\***data**;** int size**;**

ui**->**action\_clients**->**trigger**();**

**for(**int i**=**0**;**i**<**hSlots**.**size**();++**i**){**

**if(**ReceiveData**(**hSlots**[**i**],**data**,**size**,\***states**[**i**])){**

QBuffer buffer**;**

QPixmap picture**;**

buffer**.**setData**(**QByteArray**::**fromRawData**(**data**,**size**));**

buffer**.**open**(**QIODevice**::**ReadOnly**);**

QDataStream in**(&**buffer**);**

in**>>**picture**;**

buffer**.**close**();**

scenes**[**i**]->**clear**();**

scenes**[**i**]->**addPixmap**(**picture**);**

**delete** data**;**

**}**

**}**

**}**

void MainWindow**::**on\_action\_clients\_triggered**()**

**{**

QUuid uuid**;** HANDLE hSlot**;**

**if(**ReceiveNotify**(**hNotify**,**uuid**)){**

qDebug**()<<**uuid**.**toString**();**

**if(**uuids**.**indexOf**(**uuid**)==-**1**){**

ServerState **\***state**=new** ServerState**;**

state**->**current\_num**=**0**;**state**->**first\_run**=true;**

states**<<**state**;**

OpenUUIDSlot**(**hSlot**,**uuid**);**

uuids**<<**uuid**;**

hSlots**<<**hSlot**;**

QGraphicsView **\***gv**=new** QGraphicsView**;**

QGraphicsScene **\***gs**=new** QGraphicsScene**;**

scenes**<<**gs**;**

gv**->**setScene**(**gs**);**

ui**->**tabWidget**->**addTab**(**gv**,**uuid**.**toString**());**

**}**

**}**

**}**

void MainWindow**::**on\_tabWidget\_tabCloseRequested**(**int index**)**

**{**

CloseHandle**(**hSlots**[**index**]);** hSlots**.**removeAt**(**index**);**

uuids**.**removeAt**(**index**);**

**delete** scenes**[**index**];**scenes**.**removeAt**(**index**);**

ui**->**tabWidget**->**removeTab**(**index**);**

**}**

**Клиентское приложение.**

mainwindow.h:

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QBuffer>

#include <QPicture>

#include <QUuid>

#include "graphicsscene.h"

extern "C"**{**

#include "windows.h"

**}**

**namespace** Ui **{**

class MainWindow**;**

**}**

class MainWindow **:** public QMainWindow

**{**

Q\_OBJECT

public**:**

explicit MainWindow**(**QWidget **\***parent **=** 0**);**

**~**MainWindow**();**

private slots**:**

void on\_action\_send\_triggered**();**

void on\_action\_load\_triggered**();**

private**:**

Ui**::**MainWindow **\***ui**;**

GraphicsScene scene**;**

HANDLE hSlot**,**hNotify**;**

QUuid uuid**;**

int num**;**

**};**

#endif // MAINWINDOW\_H

mainwindow.cpp:

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "mailslots.h"

#include <QDebug>

#include <QPixmap>

#include <QFileDialog>

MainWindow**::**MainWindow**(**QWidget **\***parent**)** **:**

QMainWindow**(**parent**),**

ui**(new** Ui**::**MainWindow**)**

**{**

num**=**0**;**

ui**->**setupUi**(this);**

ui**->**graphicsView**->**setScene**(&**scene**);**

uuid**=**QUuid**::**createUuid**();**

MakeFile**(**hNotify**,**PutServerNotify**);**

SendNotify**(**hNotify**,**uuid**);**

//ui->graphicsView\_2->setAttribute(Qt::WA\_OpaquePaintEvent);

**}**

MainWindow**::~**MainWindow**()**

**{**

**delete** ui**;**

**}**

void MainWindow**::**on\_action\_send\_triggered**()**

**{**

QBuffer buffer**;**

int w**=**scene**.**width**()+**1**,**h**=**scene**.**height**()+**1**;**

QRectF src**=**scene**.**sceneRect**(),**dst**;**

dst**.**setBottomLeft**(**QPoint**(**0**,**0**));**

src**.**setWidth**(**w**);** src**.**setHeight**(**h**);**

dst**.**setWidth**(**w**);** dst**.**setHeight**(**h**);**

QPixmap picture**(**w**,**h**);**

picture**.**fill**(**QColor**(**"white"**));**

QPainter painter**(&**picture**);**

scene**.**render**(&**painter**,**dst**,**src**);**

painter**.**end**();**

buffer**.**open**(**QIODevice**::**WriteOnly**);**

QDataStream out**(&**buffer**);**

out**<<**picture**;**

buffer**.**close**();**

qDebug**()<<**"Sending:"**<<**buffer**.**size**();**

GenUUIDSlot**(**hSlot**,**uuid**);**

TransmitData**(**hSlot**,**buffer**.**data**().**constData**(),**buffer**.**data**().**size**(),**num**);**

CloseHandle**(**hSlot**);**

**}**

void MainWindow**::**on\_action\_load\_triggered**()**

**{**

QString name**=**QFileDialog**::**getOpenFileName**();**

**if(**name**!=**""**){**

QPixmap pixmap**;**

pixmap**.**load**(**name**);**

scene**.**clear**();**

scene**.**bg**=**scene**.**addPixmap**(**pixmap**);**

**}**

**}**

# Тестовый пример

На рисунке 1 представлен пример работы комплекса программ для рисования и передачи изображений.

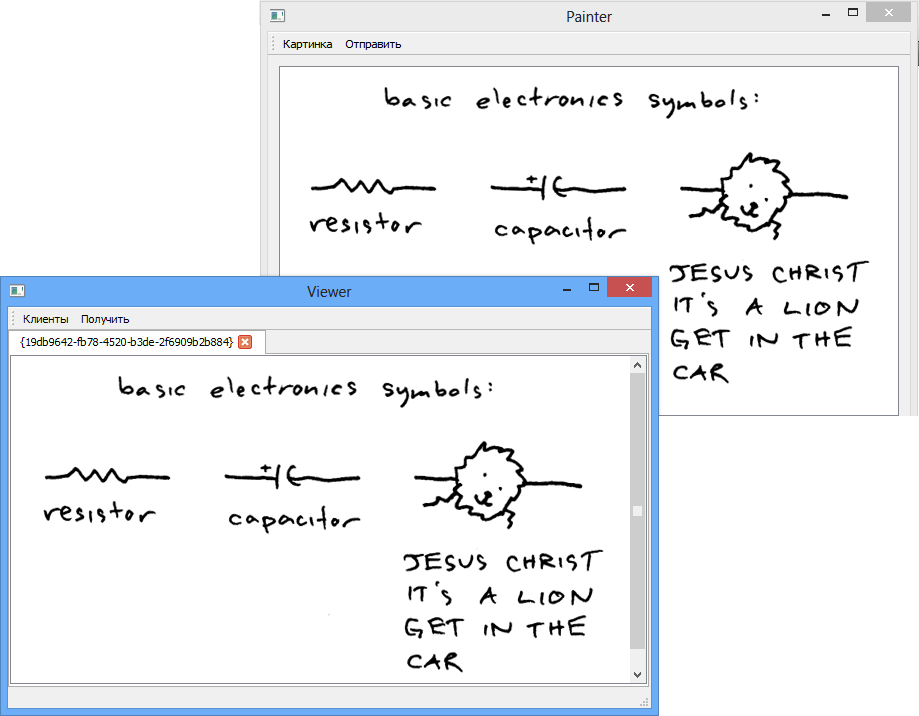


Рисунок 1— Пример работы программ

# Вывод

MailSlot – механизм межпроцессного взаимодействия на основе дейтаграмм. К плюсам можно отнести высокую скорость и удобство работы, возможность работы по сети, в том числе и в широковещательном режиме. К недостаткам – ограничение на размер пересылаемого сообщения (416-428 байт, зависит от длины имени mailslot’а), отсутствие надежности (следствие применение дейтаграмм).